VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Bang! karetní hra

Bang! card game

2016

Dominik Pecuch

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Dominik Pecuch**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Karetní hra Bang!
Bang! Card Game

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit elektronickou verzi karetní hry Bang!. Výsledkem bude desktopová aplikace určená pro hráče a serverová aplikace, která bude průběhy her řídit. Součástí bude rovněž databáze s informacemi o uživatelích (hráčích), hrách, turnajích, výsledcích a jednotlivých tazích.

Práce bude obsahovat následující body:
1. Stručný popis pravidel hry.
2. Rešerše stávajících řešení.
3. Návrh architektury řešení.
4. Návrh síťového protokolu pro komunikaci mezi herní aplikací a serverem.
5. Návrh struktury databáze.
6. Návrh a implementace herní aplikace.
7. Návrh a implementace herního serveru.
8. Srovnání s existujícími implementacemi.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Lukáš**

Datum zadání: 01.09.2014
Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Čestné prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, z kterých jsem čerpal.

V Ostravě 29. duben 2016

.....

Rád by jsem poděkoval panu Ing. Petru Lukášovi za odborné vedení a rady při vypracování této bakalářské práce.

Abstrakt

Cílem bakalářské práce je navrhnout a implementovat karetní hru Bang!. První část se bude zabývat serverem, kde se bude zpracovávat logika hry. Druhá část bude zaměřena na implementaci rozhraní pro klienta, přes které se bude realizovat připojení na server a zobrazení průběhu hry. Třetí část zahrnuje návrh a vytvoření databáze, kde se budou ukládat výsledky her.

Klíčové slova: server, klient, implementace, framework, .NET Framework, C#, databáze

Abstract

Bachelor thesis deals with design and implementation of a Bang! card game. First part focuses on server, where the game logic is handled. Second part deals with implementation of user control interface for a player that would create a connection to the server and display the games progress. The last part describes design and creation of main database, where game score is stored.

Keywords: server, client, implementation, framework, .NET Framework, C#, database

Seznam použitých zkratek a symbolů

OS	–	Operační systém
HW	–	Hardware
TCP	–	Transmission Control Protocol
IP	–	Internet Protocol
XML	–	Xtensible Markup Language
SQL	–	Structured Query Language
ORM	–	Object-relational mapping

Obsah

1 Úvod **5**

2 Karetní hra Bang! **6**
2.1 Základní informace o hře 6
2.2 Pravidla hry 6

3 Architektura řešení **9**
3.1 Uživatelé 10
3.2 Server 11
3.3 Společná knihovna 13
3.4 Klient 22
3.5 Databáze 32

4 Předchozí řešení **36**

5 Závěr **40**

6 Reference **41**

Přílohy **42**

A Tabulka zpráv **42**

B Návod **44**

C Příloha na CD/DVD **45**

Seznam obrázků

1 Diagram rozložení hráčů . . . 6
2 Ukázka karet postav a rolí. . . . 7
3 Ukázka herních karet. . . . 8
4 Architektura systému . . . 9
5 Use case diagram práv uživatelů . . . 10
6 Třídní diagram serveru . . . 11
7 Diagram přebraní zprávy . . . 12
8 Třídní diagram rozhraní `ICards` . . . 14
9 Třídní diagram rozhraní `IRole` . . . 15
10 Třídní diagram rozhraní IPostava . . . 16
11 Třída CardsPackage . . . 16
12 Třída Hráč . . . 17
13 Třída RangeMatrix . . . 18
14 Třídní diagram herní logiky . . . 19
15 Příklad tahu . . . 20
16 Třídní diagram XML . . . 20
17 Ukázka přístupu k databázi . . . 21
18 Třídní diagram tříd pro komunikaci se serverem . . . 22
19 Příklad komunikace se serverem . . . 23
20 Formulář přihlášení a registrace . . . 24
21 Třídní diagram formuláře pro přihlášení do hry `ServerOption` . . . 25
22 Hlavní okno . . . 26
23 Třídní diagram komponenty `CardImage` . . . 27
24 Třídní diagram hlavního okna MainWindow. . . . 28
25 Výsledky. . . . 29
26 Třídní diagram formuláře třídy `Výsledky`. . . . 30
27 Administrace. . . . 30
28 Příklad třídy pro získání dat z databáze na klientovi. . . . 31
29 Relační model . . . 35
30 Ukázka hry KBang . . . 38
31 Ukázka hry Bang! Video game . . . 39
32 Server . . . 44

Seznam tabulek

1 Tabulka vzdálenosti na začátku hry 18
2 Tabulka vzdálenosti po vypadnutí hráče 18
3 Tabulka *Player* 32
4 Tabulka *Game* 32
5 Tabulka *PlayerIngame* 32
6 Tabulka *Move* 33
7 Tabulka *Event* 33
8 Tabulka *Enemies* 33
9 Tabulka *CharacterT* 34
10 Tabulka *RoleT* 34
11 Tabulka *Error_Report* 34
12 Tabulka *Chat* 34
13 Tabulka srovnání 37
14 Tabulka zpráv 42
14 Tabulka zpráv 43

Seznam výpisů zdrojového kódu

1 Ukázka metody Process: 11
2 Ukázka testování zprávy na serveru: 13
3 Ukázka inicializace ICards: 15

1 Úvod

Tato práce je zaměřena na vypracování elektronické podoby karetní hry Bang!. Aplikace je tvořena jako hra přes síť pro více hráčů.

Motivací vytvořit vlastní elektronickou verzi této hry je to, že tuto konkrétní hru znám již delší dobu v její stolní podobě a chtěl jsem jí přenést i do elektronické formy.

Cíle byly rozčleněny do bodů, které ve výsledku tvoří tuto práci, která je rozdělená na praktickou část a na teoretickou část.

Praktická část práce, bude rozdělená do čtyř bodů: společná knihovna, server, klient a databáze.

Společná knihovna bude obsahovat popis herních karet, karet postav a rolí, dále bude obsahovat definici hráče a jeho vzdálenosti od ostatních hráčů. Dále zde bude obsažená logika hry, která se bude řídit dle platných pravidel této karetní hry. Společná knihovna bude dále obsahovat přípravu složitějších zpráv zasálaných na klienta, a dále přístup do databáze, který bude moct být využit jen ze serverové části. Klient data z databáze dostane vždy jedině přes server.

Druhý bod této práce se bude zabývat návrhem serverové části. Zde se bude řešit připojení a přebírání zpráv zaslaných od klientů a následné zpracování příchozí zprávy. Na serveru bude vždy aktuální hra a bude se zde zpracovávat její logika.

Třetí bod se bude zabývat návrhem klientské části aplikace. Bude zde grafické rozhraní, které bude zprostředkovávat uživatelům hru, a komunikace se serverovou části, od které bude získávat aktuální informace týkající se právě probíhající hry. Budou zde zobrazena data z databáze, která budou získána od serveru. Klient nebude mít přímý přístup k databázi.

Poslední bod této praktické části se bude zabývat návrhem a vytvoření databáze, do které se budou ukládat záznamy z her a informace o hráčích.

Teoretická část práce bude rozdělena do tří kapitol a to na stručný popis pravidel hry, architekturu aplikace z praktické části práce a na srovnání předchozích řešení této karetní hry.

První kapitola bude rozdělená do dvou podkapitol. V první podkapitole bude rychlé seznámení se s touto karetní hrou a v druhé podkapitole budou stručně popsána pravidla hry.

Druhá kapitola se bude zabývat popisem aplikace. Bude rozdělená do čtyř podkapitol, které budou odpovídat jednotlivým bodům praktické části práce. Tyto podkapitoly budou detailně popisovat chování jednotlivých bodů aplikace.

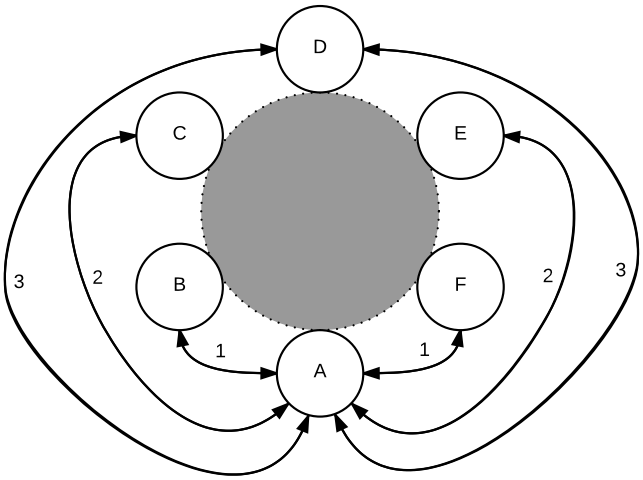
V poslední kapitole bude popsáno srovnání této práce, s některými již funkčními řešeními, které jsou v současné době dostupné.

2 Karetní hra Bang!

V této kapitole budou popsána obecná pravidla karetní hry Bang!.

2.1 Základní informace o hře

Karetní hra Bang! je strategická karetní hra na motivy divokého západu. Hra je určena pro 3 až 8 hráčů. V základním rozšíření obsahuje 7 karet rolí (role jsou karty, které určují pozdější chování ve hře), 16 karet postav (určující vlastnosti daného hráče po celý zbytek hry a jeho počet životů) a 80 herních karet. Herní karty jsou navíc označeny hodnotami, jaké mají běžné karty: 2-3-4-5-6-7-8-9-10-J-Q-K-A a v kombinaci s: srdce, piky, káry, kříže. Herní karty jsou navíc dvojího druhu, karty s modrým okrajem, které se vykládají na stůl před hráče a přidávají mu další vlastnosti a karty s hnědým okrajem, které má hráč v ruce a používá je ke hře. Rozsazení u stolu je důležité v případě, že se jedná o hru, pro víc jak 3 hráče, neboť se zde řeší vzdálenosti od daného hráče k dalšímu v případě, že je vzdálen víc jak o 1, jak můžeme vidět na Obrázku 1.



Obrázek 1: Diagram rozložení hráčů

2.2 Pravidla hry

V této části budou stručně popsána pravidla této karetní hry.

2.2.1 Role

Cíl každého hráče záleží na roli, kterou si na začátku hry vylosuje (viz Obrázek 2a).
Šerif: musí zabít všechny zločince a odpadlíka.
Banditi: musí zabít šerifa dříve, než on zabije je.

Pomocníci: pomáhají šerifovi a vyhrají, jen když vyhraje on.
Odpadlík: chce se stát novým šerifem, proto musí zabít šerifa jako posledního a zůstat jako poslední hráč ve hře.

2.2.2 Postavy

Každá postava (např. Obrázek 2b) má unikátní vlastnosti, které může hráč využívat po celou hru. Tyto schopnosti ovlivňují následující hru a strategii daného hráče. O tyto schopnosti nemůže hráč v průběhu hry nikdy přijít, navíc daná schopnost nemusí být využita, když daný hráč nemá zájem. Nábojnice na postavě znázorňují počet životů, které daný hráč bude mít. V případě že daný hráč bude šerif, bude mít +1 život navíc. Počet životů na konci kola také určuje počet karet, které daný hráč může mít v ruce na konci svého tahu.
Příklad karet (viz Obrázek 2):

(a) Karta role

(b) Karta postavy

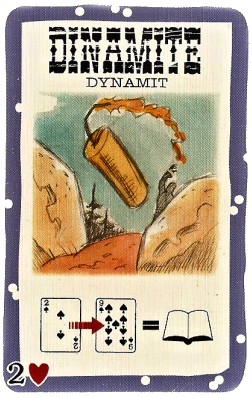
Obrázek 2: Ukázka karet postav a rolí.

2.2.3 Herní karty

Herní karty jsou dvojího druhu, jedny, které přidávají vlastnosti hráči (karty s modrým okrajem viz Obrázek 3a) a samotné herní karty, které má hráč v ruce (karty s hnědým okrajem viz Obrázek 3b). Karty s modrým okrajem se vykládají na stůl, teprve pak hráč získává dočasně jejich vlastnost, tyto karty mohou být odstraněny jinými hráči. Karty s hnědým okrajem se využívají na herní akce.

2.2.4 Zahájení hry

Při začátku hry si hráči nejprve vylosují role, poté postavy, pak se rozdají karty (každému 5 karet do začátku). Hru začíná šerif, dále se pokračuje ve směru hodinových ručiček, až se dojde zpátky k hráči s rolí šerifa. Každé kolo daného hráče začíná sejmutím 2 karet z balíčku, nato pokračuje zahráním herních karet popřípadě vyložením karet na stůl. Na



(a) Příklad karty s modrým okrajem

(b) Příklad karty s hnědým okrajem

Obrázek 3: Ukázka herních karet.

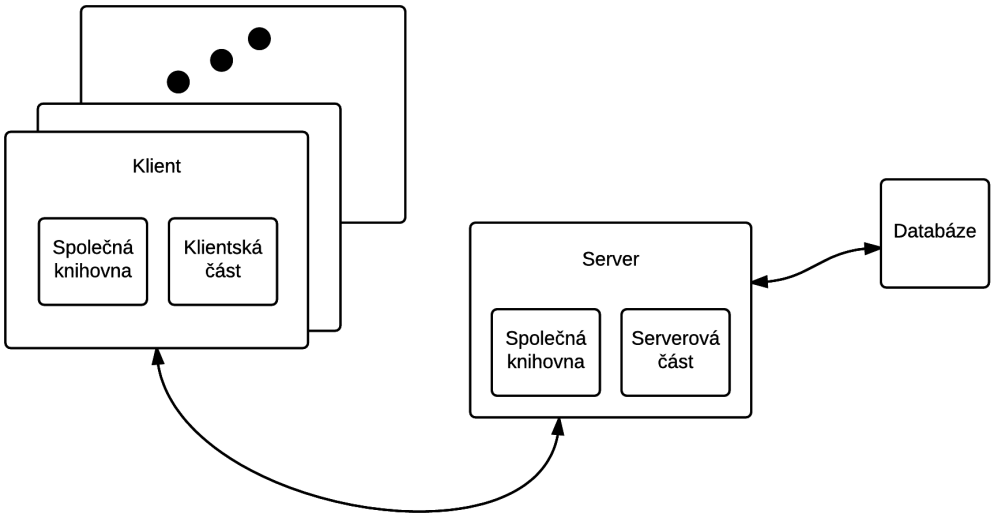
konci tahu musí mít hráč v ruce tolik karet, kolik má aktuálně životů, například v prvním tahu byl hráči A odečten život, v dalším kole si hráč A život neobnovil a má 3 životy, čili na konci svého tahu musí mít pouze 3 karty v ruce. Proto na konci svého tahu hráči odhazují nadbytečné karty.

2.2.5 Konec hry

Hra končí v různých případech.

- Šerif umřel, protože ho zabil bandita, bandité vítězí.
- Šerif umřel, neboť ho zabil odpadlík, vyhrává odpadlík.
- Umřeli bandité a odpadlík, vyhrává šerif.

3 Architektura řešení



Obrázek 4: Architektura systému

Architekturu této hry je možné rozdělit na čtyři části, viz Obrázek 4:

1. Společná knihovna
2. Klient
3. Server
4. Databáze

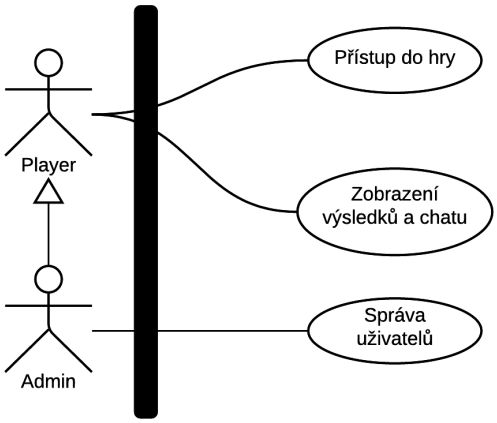
Společná knihovna (viz kapitola 3.3) je důležitý základ celé aplikace. Obsahuje definici herních karet, karet rolí a postav, dále je zde i vytvoření kolekcí všech karet. Tato knihovna zde obsahuje třídu s logikou hry a třídy pro přístup do databáze.

Na serveru (viz kapitola 3.2) se vytváří hra ze společné knihovny a dochází zde k přístupu do databáze přes společnou knihovnu. Server přebírá zprávy od klienta, podle kterých spustí příslušnou funkcionalitu z logiky hry.

Klient (viz kapitola 3.4) řeší zobrazování hry a výsledků z databáze. Veškeré informace jsou uložené na serveru a klient si je podle potřeby vyžádá a zobrazí.

Do databáze (viz kapitola 3.5) jsou ukládány veškeré výsledky, záznamy o tazích, chatu a informace o hráčích. Klient nemá přímé spojení na databázi a informace si vyžaduje přes server, který mu pošle příslušné záznamy.

3.1 Uživatelé



Obrázek 5: Use case diagram práv uživatelů

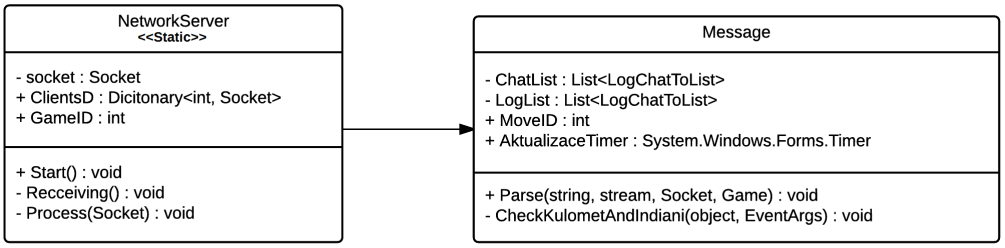
Tato aplikace je víceuživatelský systém. Uživatelé mají dva typy práv (viz Obrázek 5):

- administrátor,
- hráč.

V aplikaci může administrátor přistupovat do správy uživatelů a přehledu chyb v aplikaci, dále má možnost ve výsledcích zobrazit i ostatní hráče a chat, který vedli.

Hráč do administrace nemá přístup. Ve výsledcích vidí pouze hry, které sám odehrál, a chat, který sám poslal.

3.2 Server



Obrázek 6: Třídní diagram serveru

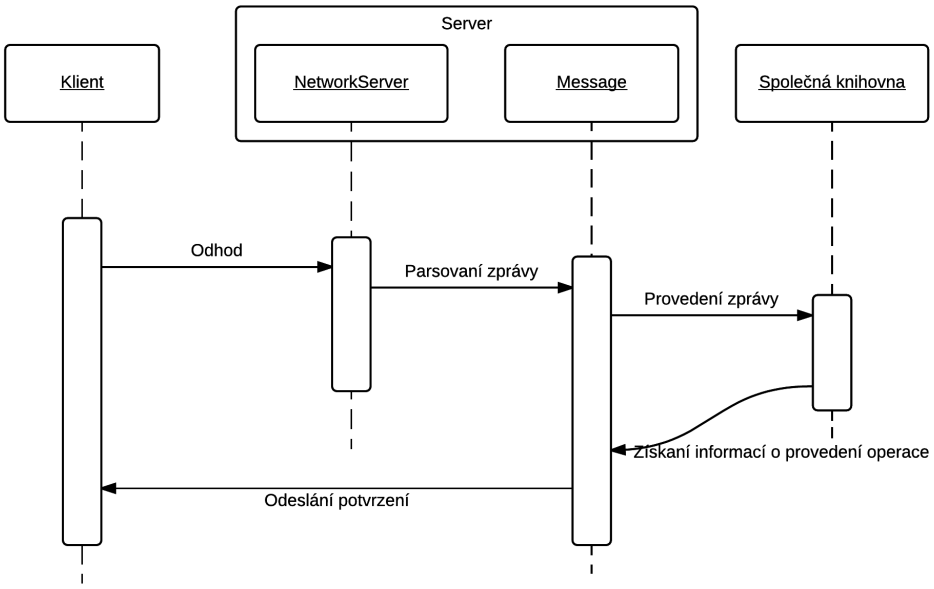
Server se skládá ze dvou základních tříd `NetworkServer` a `Message` (viz Obrázek 6).

Statická třída `NetworkServer` udržuje kolekci klientů ve slovníku, kde klíčem je id klienta a hodnotou socket, tj. spojení s klientem. Klienta zaregistruje v metodě `Reciving`, kde se pro každého klienta vytvoří speciální vlákno pro metodu `Process` (viz Výpis 1), kde přebírá zprávy. Tato metoda převezme zprávu od klienta a následně ji deleguje do instance třídy `Message` a její metody `Parse`, kde se tato zpráva parsuje podle prvního slova a poté se spustí příslušná metoda.

```
1  if (client == null) return;
2  string message = "";
3  do{
4     try{
5        Stream stream = new NetworkStream(client);
6        var streamReader = new StreamReader(stream);
7        message = streamReader.ReadLine();
8        MessSend.Parse(message, stream, client, g);
9     }
10    catch (Exception e){
11       if (client.Connected == false)
12       message = "q";
13    }
14 } while (!"q".Equals(message));
15 client.Close();
```

Výpis 1: Ukázka metody Process:

Na řádku 1 ve Výpise 1 se otestuje zda připojený klient nenabývá hodnoty `null`. Dále v cyklu na řádcích 3-17 se čte zpráva zaslaná klientem viz řádek 9 a následně se deleguje do instance třídy `Message` viz řádek 10. Pokud nastane chyba a klient nebude nadále připojen na serveru cyklus se pro něj ukončí a server klienta odpojí.



Obrázek 7: Diagram přebraní zprávy

3.2.1 Formát předávání zpráv

Mezi klientem a serverem se vyměňují zprávy ve formátu:

`{ [Hlavička], [parametr 1], . . . ,[parametr n] }`.

Hlavička ve zprávě značí, jaká metoda se má provést, a parametry se následně využívají v dané metodě. Ve zprávě se hlavička a parametry oddělují čárkou. Parsování zprávy z jednoho řetězce se provádí pomocí čárky, kdy se zpráva rozdělí na jednotlivá slova, která se uloží do pole, kde první prvek je hlavička, podle které se provede příslušná metoda, do níž je odesláno pole s parametry. Zpáteční zpráva vždy obsahuje hlavičku příchozí zprávy, aby na klientovi proběhlo ověření, že se provedla správná metoda. Pokud zpráva obsahuje složitější informace, jako je například kolekce, přepošle formou XML (viz kapitola 3.3.5).

Příklad 3.1 (viz Obrázek 7)
Klient pošle zprávu na server `"Odhod, 2, Nick"`, na serveru si zprávu přebere třída `NetworkServer` a následně ji přepošle do instance třídy `Message`. V této instanci třídy, v metodě `Parse`, se zpráva rozparsuje a načež se uloží do pole. Poté se první slovo testuje ve switchi (viz Výpis 2), po nalezení správné podmínky se provede metoda `Odhaz`. Po provedení se odešle zpráva zpátky na klienta, v ní bude jen hlavička `Odhod`.

```
1  string [] tmp = message.Split(',');
2
3  switch (tmp[0])
4  {
5     case "NewGame":
6     NewPlayer(streamWriter, g, tmp);
7     break;
8     case "AllPlayers":
9     AllPlayers(streamWriter, g, tmp);
10    break;
11    case "Health":
12    Health(streamWriter, g, tmp);
13    break; ...
```

Výpis 2: Ukázka testování zprávy na serveru:

3.2.2 Protokol komunikace

Tento navržený protokol funguje na principu zpráva-odpověď. Formát zprávy je popsán v kapitole 3.2.1.

Zahájení komunikace přichází vždy od klienta, který zašle zprávu na server a čeká na příchozí odpověď. Zpráva vždy začíná tzv. hlavičkou zprávy, kde je definováno, jaký typ operace se má provést. Může pokračovat i parametry, pokud si to daná operace vyžaduje. Po obdržení zprávy na serveru a provedení příslušné operace se naformuluje příslušná zpráva odpovědi. Musí mít stejnou hlavičku, jako měla příchozí zpráva, jelikož podle této se ověřuje, že se provedla správná operace na serveru. Může obsahovat i parametry pokud si to klient vyžaduje. V parametrech zpětných zpráv se může posílat více kolekcí, vzhledem k tomu, že se posílá jen řetězec, zabalí se kolekce do XML (viz kapitola 3.3.5) a nato se zašle zpátky na klienta.

Veškeré zprávy jsou popsány v příloze A Tabulce 14.

3.3 Společná knihovna

Knihovna je část aplikace, která obsahuje definice herních karet, karet postav a rolí. Dále obsahuje herní logiku, přístup k databázi, vytvoření zprávy do XML, definice hráče a definice matice, která určuje vzdálenost mezi hráči.

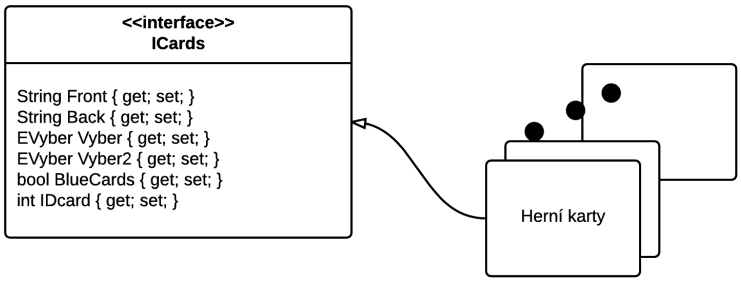
3.3.1 Karty

Karty jsou nedílnou součástí této hry, proto se musí určit, jak budou vypadat a jak se budou projevovat. Karty se dělí na tři skupiny:

- herní karty,
- karty rolí,
- karty postav.

Jednotlivé skupiny jsou implementovány samostatně a jsou popsány v následujících třech podkapitolách.

3.3.1.1 Herní karty



Obrázek 8: Třídní diagram rozhraní `ICards`

První skupina karet implementuje rozhraní `ICards` (viz Obrázek 8). Toto rozhraní obsahuje šest vlastností:

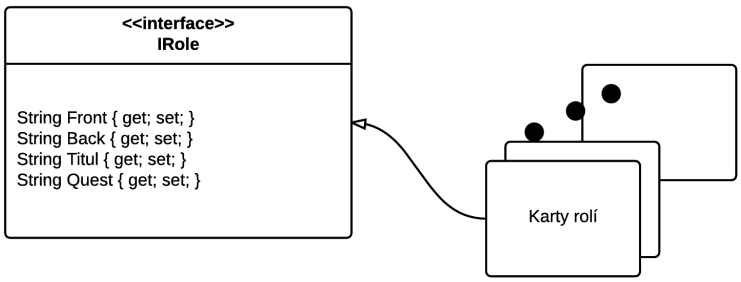
- `Front` – Přední strana karty. Cesta k obrázku.
- `Back` – Zadní strana karty. Cesta k obrázku.
- `Vyber` – Výčet vlastností karty, zde jsou uvedeny tyto hodnoty: 2-3-4-5-6-7-8-9-10-J-Q-K-A
- `Vyber2` – Výčet vlastností, kde jsou uvedeny tyto hodnoty: Piky, Káry, Srdce, Kříže
- `BlueCard` – Příznak, zda daná karta je s modrým okrajem či nikoli.
- `IDcard` – Unikátní identifikátor karty.

Při vytvoření karty je předem nastavený příznak, jaká karta bude s modrým okrajem a jaká ne. Jediné, co se nastavuje při vytvoření karty, jsou cesty k obrázkům, nastavení výběrů a ID karty. Příklad vytvoření karty viz Výpis 3

```
1  Bang bang1 = new Bang(EVyber.krize, EVyber2.sest, 1,
       Path.Combine(path, "Bang6krize.png"),
       Path.Combine(path,"BangBack.png"));
```

Výpis 3: Ukázka inicializace ICards:

3.3.1.2 Karty rolí

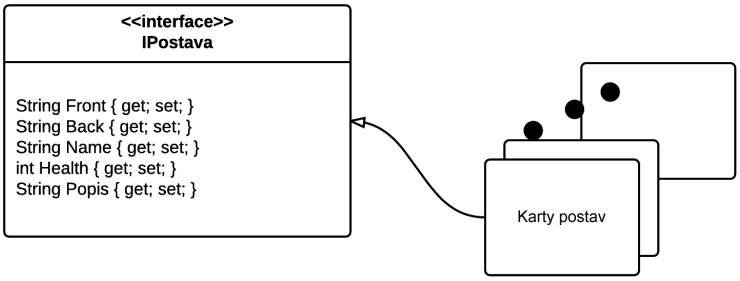


Obrázek 9: Třídní diagram rozhraní `IRole`

Druha skupina karet rolí implementuje rozhraní `IRole` (viz Obrázek 9), které obsahuje dvě stejné vlastnosti pro cesty k obrázkům jako v rozhraní `ICards`. Další vlastnosti u tohoto rozhraní jsou titul (`Titul`) a úkol (`Quest`). Při vytváření jakékoli role jsou všechny následující vlastnosti nastaveny a nemusí se tak vyplňovat další parametr. Tyto vlastnosti jsou:

- `Front` – Přední strana karty. Cesta k obrázku.
- `Back` – Zadní strana karty. Cesta k obrázku.
- `Viditelnost` – Udává, zda daná role může být zobrazena hráčům. Jen role šerifa lze vidět.
- `Titul` – Udává, jak se role jmenuje.
- `Quest` – Popis, co má role vykonat během hry.

3.3.1.3 Karty postav

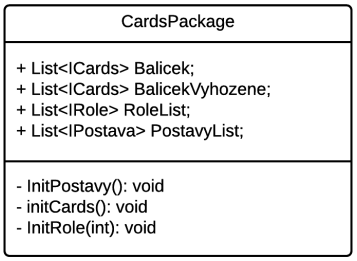


Obrázek 10: Třídní diagram rozhraní IPostava

Karty postav představují poslední skupinu karet, jež implementují rozhraní `IPostava`, viz Obrázek 10, které opět obsahují dvě stejné vlastnosti pro cesty k obrázkům, jako v rozhraní `ICards`. Obsahují mimo jiné své jméno, počet životů jako i popis schopností postavy. Tyto vlastnosti jsou:

- `Front` – Přední strana karty. Cesta k obrázku.
- `Back` – Zadní strana karty. Cesta k obrázku.
- `Name` – Jméno postavy.
- `Health` – Počet životů postavy.
- `Popis` – Popis vlastností postavy.

3.3.1.4 Balíčky



Obrázek 11: Třída CardsPackage

Ve třídě `CardsPackage`, viz Obrázek 11, se vytváří instance tříd karet, které jsou zmíněny v předchozích podkapitolách, jsou postupně přidány do kolekcí, které jsou před začátkem každé hry promíchány.

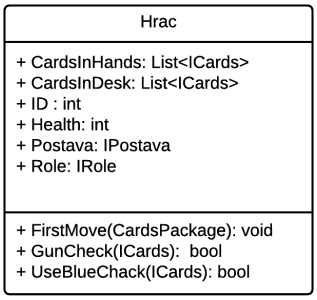
Pro herní karty jsou vytvořeny dvě kolekce, první kolekce `Balicek` obsahuje promíchané karty pro hru a druhá kolekce `BalicekVyhozene` obsahuje použité karty, které se odhodily během hry. Tato druhá kolekce je během hry neustále kontrolována, karty v ní jsou přesouvány do první kolekce po překročení limitu pěti karet, pro zajištění dostatku karet v herní kolekci.

Karty rolí se vytváří na základě počtu hráčů ve hře dle platných pravidel hry (viz kapitola 2). Po inicializaci hry se karty rolí rozdělí mezi hráče a z této kolekce se odstraní.

Karty postav se vytvoří a přidají do kolekce stejně, jak herní karty, ale s tím rozdílem, že po inicializaci hry se rozdělí každému hráči náhodně jedna postava. Dále se již tato kolekce ve hře nevyužívá.

Server i klient mají svou instanci této třídy. Na serveru je vždy platná verze balíčku. Klient pravidelně posílá požadavky o synchronizaci svého balíčku.

3.3.2 Hráč



Obrázek 12: Třída Hráč

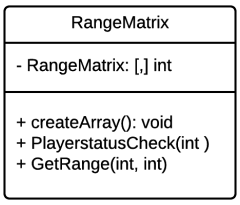
Třída `Hrac` (viz Obrázek 12) je důležitou součástí této aplikace. Obsahuje unikátní ID hráče, jeho nick, životy, kolekci karet na stole, kolekci karet v ruce a další informace o hráči. Dále je zde funkce `FirstMove`, která přidá, na začátku každého tahu, hráči karty do ruky.

Přemístění karet z kolekce karet v ruce do kolekce karet na stole jde jen v případě, že karta má příznak modrého okraje.

3.3.3 Vzdálenostní matice

Instance třídy `RangeMatrix` slouží k určení vzdálenosti mezi hráči. Maticí `RangeMatrix` se určuje zda aktuální hráč může použít kartu na vybraného hráče (viz kapitola 2).

Základ této třídy tvoří dvourozměrné pole, které se alokuje při vytvoření. Pokud bude hra například pro pět lidí, bude pole 5x5. Při smrti hráče se provede metoda `PlayerStatusCheck`, která v poli zmenší vzdálenost mezi hráči. Pro získání aktuální vzdálenosti se využívá metoda `GetRange`. Ta po vložení ID hráčů vrátí příslušnou vzdálenost.



Obrázek 13: Třída RangeMatrix

Příklad 3.2 (Hra pro pět hráčů.)
Na začátku hry pro pět hráčů se vytvoří pole, které bude vypadat jako Tabulka 1. Čísla v tabulce určují příslušnou vzdálenost mezi hráči. Během hry, vypadne hráč 3, následně se v tabulce na jeho pozici vyplní -1 (viz Tabulka 2), která značí, že daný hráč už nehraje, poté se přepočítají vzdálenosti a upraví se v tabulce.

Tabulka 1: Tabulka vzdálenosti na začátku hry

	Hráč 1	Hráč 2	Hráč 3	Hráč 4	Hráč 5
Hráč 1	0	1	2	2	1
Hráč 2	1	0	1	2	2
Hráč 3	2	1	0	1	2
Hráč 4	2	2	1	0	1
Hráč 5	1	2	2	1	0

Tabulka 2: Tabulka vzdálenosti po vypadnutí hráče

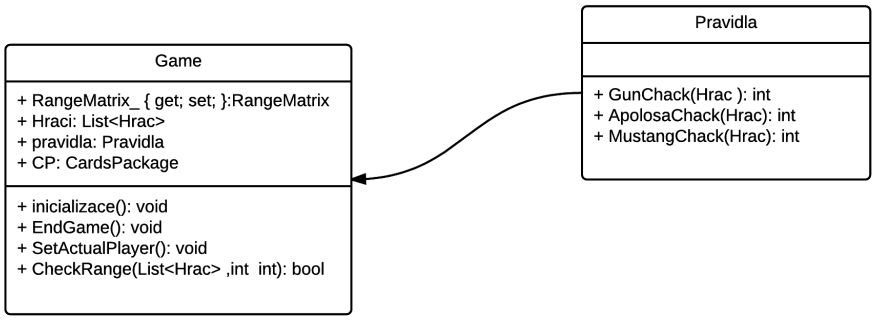
	Hráč 1	Hráč 2	Hráč 3	Hráč 4	Hráč 5
Hráč 1	0	1	-1	2	1
Hráč 2	1	0	-1	1	2
Hráč 3	-1	-1	0	-1	-1
Hráč 4	2	1	-1	0	1
Hráč 5	1	2	-1	1	0

3.3.4 Herní logika

Tato část aplikace zajišťuje veškerou logiku ve hře, zde se také vytváří celá hra. Skládá se ze tří tříd: `Game, PravidlaPostav, Pravidla`.

Třída `PravidlaPostav` je statická a využívá se jen při použití vlastnosti dané postavy, pokud samozřejmě nemá postava vlastnost, která se provádí automaticky.

Při spuštění aplikace se vytvoří instance třídy `Game`, viz Obrázek 14, ve které se zpracovávají veškeré herní akce. Při přihlášení hráče na server se vytvoří instance třídy `Hrac`



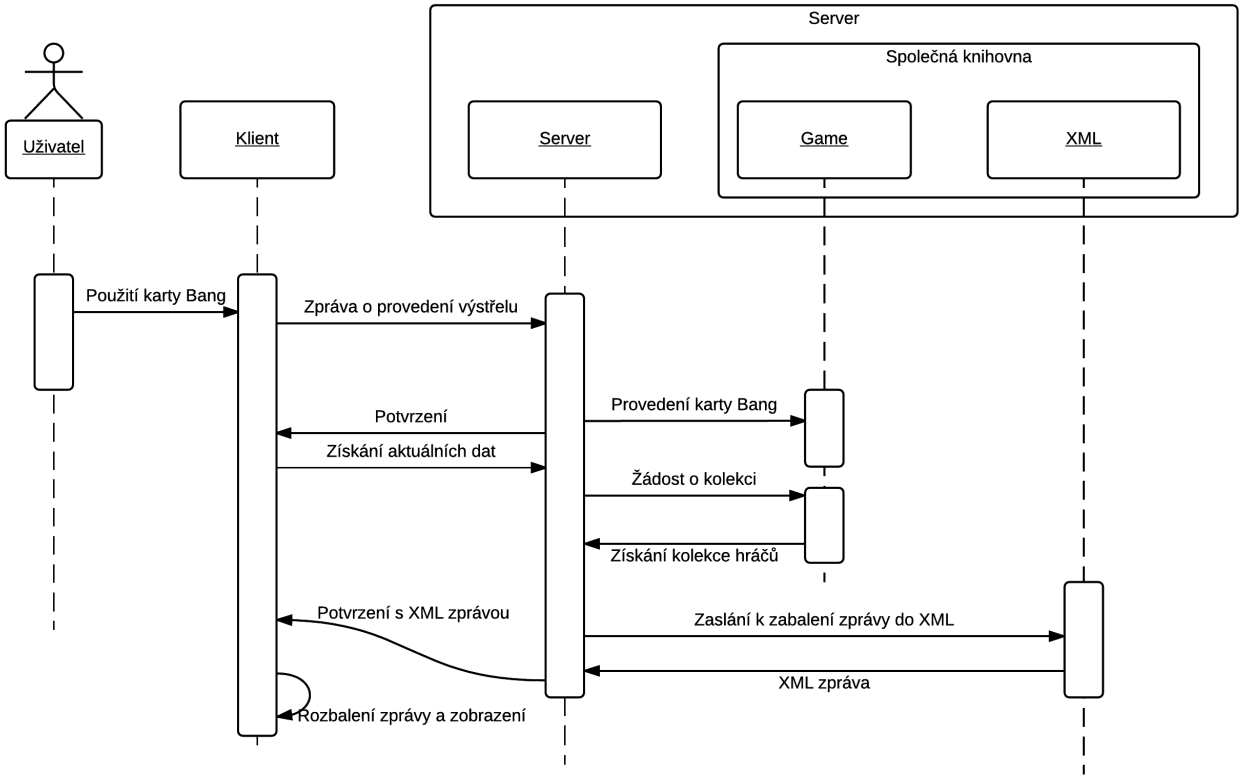
Obrázek 14: Třídní diagram herní logiky

a z instance `CardsPackage` mu přidělí roli a postavu (viz podkapitola 3.3.2), která se poté přidá do kolekce `Hraci`. Pokud jsou na serveru již všichni hráči, provede metoda `inicializace`, ta rozdá karty všem hráčům v kolekci.

Rozdání herních karet probíhá následovně. Z instance třídy `CardsPackage`, z kolekce `Balicek`, se vybere prvních pět karet, které se z tohoto balíčku odeberou a přiřadí se do kolekce příslušného hráče `CardInHands`. Toto se opakuje pro všechny hráče v kolekci. Kolekce `Hraci` se využívá na klientovi i na serveru. V této kolekci je uložen aktuální stav hráčů ve hře. Během hry jsou v této instanci využívány různé metody k právě zahraným kartám, které se projeví v příslušné instanci `Hrac` změnou životů a karet v balíčcích. Na konci tahu hráče se provede metoda `EndGame`, ta nastaví dalšímu hráči v kolekci možnost hry, a aktuálnímu hráči ji znepřístupní.

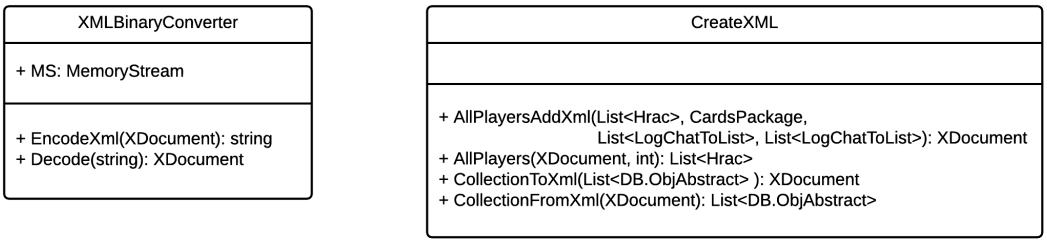
Instance třídy `Pravidla` se využívá pouze v instanci třídy `Game`. Tato třída slouží pouze na ověření různých pravidel během hry. Například, při ověřování vzdálenosti při výstřelu, se při jejím přepočtu ověřuje, zda daný hráč má vyložené karty, které zvětšují nebo zmenšují vzdálenost, a podle dané karty vrátí číslo vzdálenosti, které se buď přičte nebo odečte.

Příklad 3.3 (viz Obrázek 15)
Hráč použije kartu Bang. Z klienta se na server odešle tato zpráva `UseCard, id karty, id hráče, id nepřátelského hráče`. Na serveru se rozparsuje a v instanci třídy `Game` se provede příslušná metoda, která nepřátelskému hráči ubere život. Server zašle na klienta potvrzení o vykonání jeho požadavku. Klient pro zaktualizování hry zašle automaticky další zprávu, kterou si vyžádá aktuální data. Server po přebrání zprávy si vyžádá kolekci hráčů a následně ji pošle k zabalení do XML zprávy (viz kapitola 3.3.5). Tuto zprávu i s potvrzením odešle zpátky na klienta, který ji rozbalí a následně zobrazí.



Obrázek 15: Příklad tahu

3.3.5 XML



Obrázek 16: Třídní diagram XML

Některé zprávy vyžadují strukturované parametry v odpovědi. K tomu lze využít strukturu XML, které se posílají přes server klientovi.

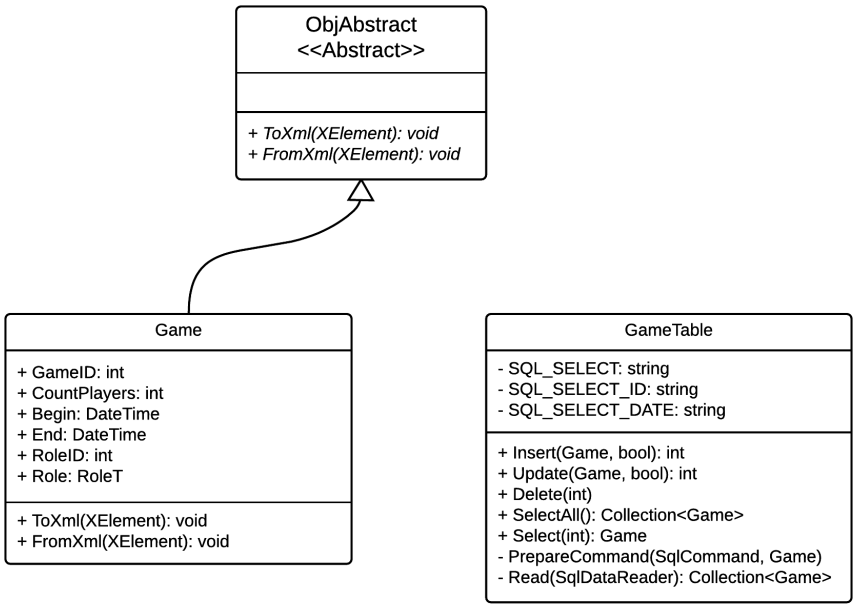
Pro vytvoření zpráv slouží instance třídy `CreateXML`, která obsahuje několik metod pro různé zprávy. Výsledkem je vždy XML dokument typu `XDocument`. Tento dokument

se pak zpracovává pomocí metod v instanci třídy `XMLBinaryConverter`. Metoda `EncodeXML` převezme dokument, posléze jej uloží do `memoryStreamu`, převede na pole bajtů a nakonec ho konvertuje na string řetězec. Metoda `Decode`, zase na druhou stranu přebírá string řetězec a z něj získá daný XML dokument.

Příklad 3.4
Klient požádá server o aktuální stav hráčů, ten vytvoří instanci třídy `CreateXML`, využije z této instance metodu `AllPlayersAddToXml`, do které zašle příslušné kolekce hráčů, logu, chatu a instanci třídy `CardsPackage`. Nato z těchto kolekcí vytvoří XML dokument, který pošle do metody `Encode` v instanci třídy `XMLBinaryConverter`, a tento řetězec zašle zpátky na klienta jako odpověď na zaslanou zprávu, jenž pomocí metody `Decode` získá XML dokumentke zpracování.

Pro názorný příklad se může uvést ukázka z předchozí kapitoly(viz Obrázek 15)

3.3.6 Přístup k databázi



Obrázek 17: Ukázka přístupu k databázi

Pro přístup k databázi se využívá ORM dle návrhového vzoru *Table data gateway*, jak uvádí Martin Fowler v [8].

Každé tabulce odpovídají dvě třídy, jedna definuje, jak tabulka vypadá v databázi, příklad třídy `Game`, viz Obrázek 17, a druhá třída obsahuje načítání dat, viz třída `GameTable` v obrázku 17. Jedna instance třídy `Game` odpovídá jednomu záznamu v tabulce v databázi.

Přístup k datům z databáze probíhá po vyvolání příslušné funkce z instance třídy `(tabulka)Table`, pro příklad `GameTable`. Pro získání dat a jejich doručení uživateli se využívají metody, které vrátí buď jednu instanci, příklad instance třídy třídy `Game`, nebo více instancí v kolekci.

Každá z metod, které přistupují k databázi, musí mít definován příslušný SQL dotaz, ať už pro získání dat, nebo jejich úpravu. Dále může vyžadovat parametry, které se pro daný dotaz připraví v metodě `PrepareCommand`. Při získávání dat přibude k funkcionalitě i metoda `Read`, která připraví instance třídy `Game`.

Pro přesun takto získaných dat přes server na klienta, slouží zabalení zprávy pomocí XML, viz kapitola 3.3.5. Pro zabalení či rozbalení dat slouží, v každé instanci těchto tříd, metody `ToXml` a `FromXml`.

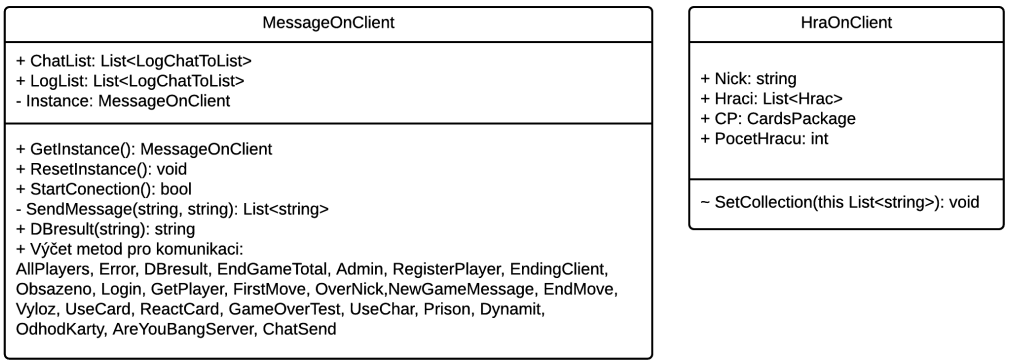
3.4 Klient

V této části aplikace dochází k veškeré interakci uživatele s hrou. Je zde grafické rozhraní, které uživateli zobrazuje průběh hry. Důležitou součástí je komunikace se serverem, díky které má uživatel vždy aktuální data a bez které by se klient ani nepustil. Klienta tedy můžeme rozdělit na tři části:

1. komunikace se serverem,
2. grafické rozhraní,
3. přístup k datům z databáze přes server.

Jednotlivé části jsou popsány v následujících třech podkapitolách.

3.4.1 Komunikace se serverem



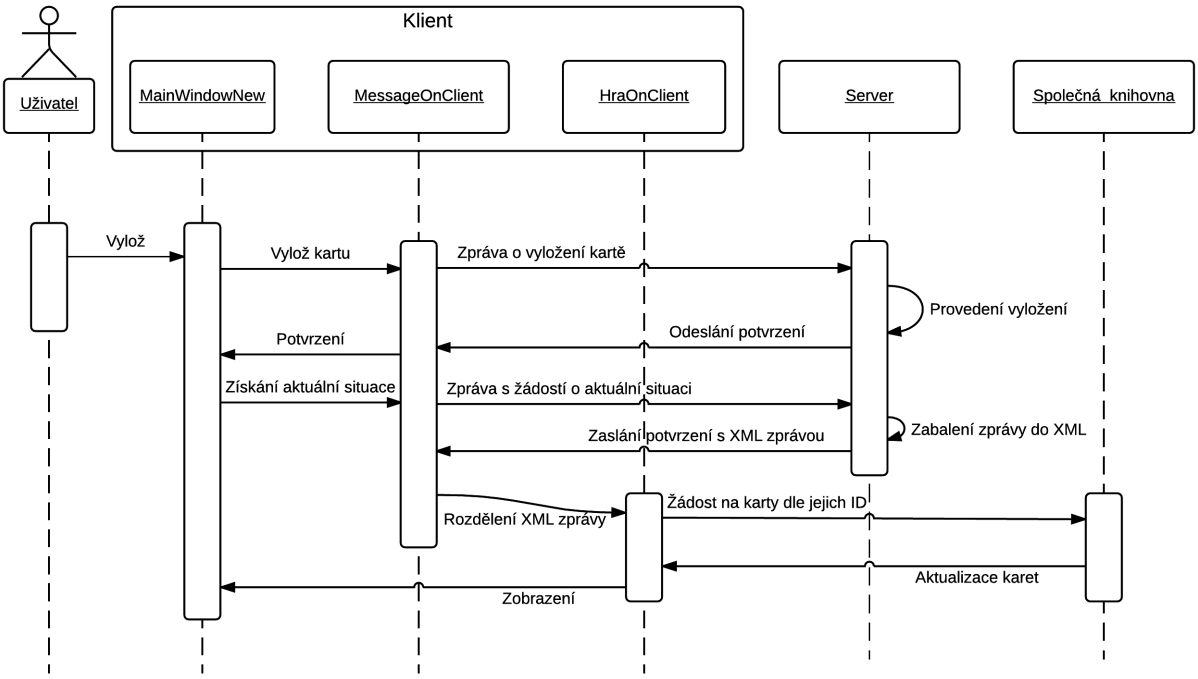
Obrázek 18: Třídní diagram tříd pro komunikaci se serverem

Třída `HraOnClient`, viz Obrázek 18, je statická třída, která udržuje aktuální kolekci a počet hráčů získaných ze serveru a instanci třídy `CardsPackage`, která reprezentuje herní

balíček (viz kapitola 3.3.1 a její podkapitola Balíček). Nachází se zde metoda `SetCollection`, ta nejdříve přebere rozparsovanou zprávu v kolekci řetězců. Zakódovaný parametr v kolekci převede na XML dokument a rozdělí jeho data do kolekcí hráčů, logu a chatu.

Třída `MessageOnClient` (viz Obrázek 18) je navržena jako singelton, jelikož se k ní přistupuje z různých částí klienta. Přes instanci této třídy se připojuje k serveru metodou `StartConection`.

Komunikace probíhá následovně. Klient vytvoří událost, buď automaticky, např. časovačem, nebo reakcí uživatele, např. použití karty. Zavolá se příslušná metoda, která v některých případech vyžaduje parametry, jako je například id uživatele, nebo jeho nick. Dle této metody dojde k sestavení hlavičky zprávy, viz kapitola 3.2.1, jenž je následně předána metodě `SendMessage` zajišťující komunikaci se serverem pomocí protokolu TCP/IP, a přebírá dvě proměnné: hlavičku zprávy a případně parametry zprávy. Po obdržení odpovědi ze serveru dojde k jejímu rozparsování a k otestování o tom, že vše proběhlo v pořádku. Hlavička odpovědi se musí shodovat s hlavičkou zprávy (viz Kapitola 3.2.1)



Obrázek 19: Příklad komunikace se serverem

Příklad 3.5 (viz Obrázek 19)
Uživatel vyloží kartu v hlavním okně `MainWindow`. Po stisku tlačítka *Vylož* dojde k sestavení řetězce obsahujícího id hráče a id karty. Tento řetězec se zašle do metody `Vyloz` v instanci třídy `MessageOnClient`, kde se k tomuto řetězci s parametry přidělí příslušná hlavička zprávy (viz Tabulka 14), poté ji přes metodu `SendMessage` zašle na server a čeká na

odpověď. Po vykonání příslušné funkcionality na serveru vznikne odpověď, která musí obsahovat vždy stejnou hlavičku jakou server dostal v původním požadavku. Podle této hlavičky se na klientovi testuje, zda server provedl správnou funkci.

Po obdržení odpovědi je okamžitě odeslána zpráva o aktualizaci, tato metoda ovšem nevyžaduje žádné parametry, proto se na server zasílá jen hlavička. Tato konkrétní zpráva očekává, kromě hlavičky zprávy, strukturovaný výstup, proto se výsledek zabalí do XML dokumentu (viz kapitola 3.3.5). Klient po obdržení odpovědi odešle XML do metody `SetCollecton` ve statické třídě `HraOnClient`, kde XML rozbalí a aktualizuje příslušné kolekce.

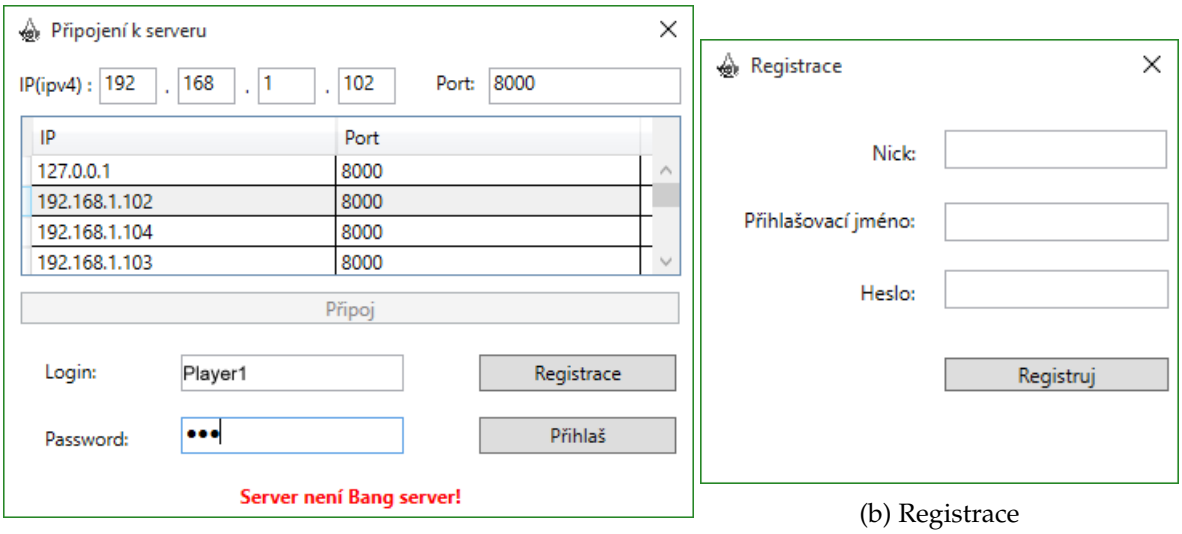
3.4.2 Grafické rozhraní

Grafické rozhraní lze dále rozčlenit na čtyři části:

1. přihlášení,
2. hlavní okno,
3. zobrazení výsledků,
4. administrace.

Jednotlivé části jsou popsány v následujících podkapitolách.

Přihlášní



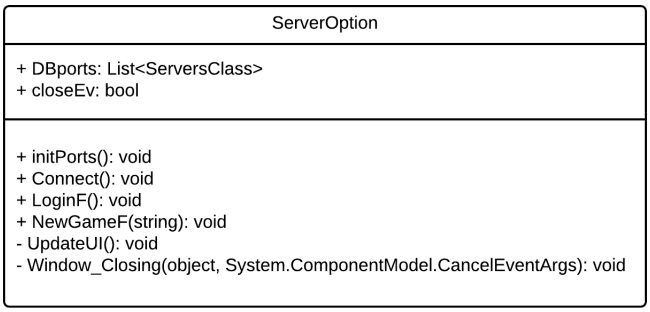
(a) Přihlášení

(b) Registrace

Obrázek 20: Formulář přihlášení a registrace

Po spuštění aplikace se objeví okno pro připojení a přihlášení na server, viz Obrázek 20a. Nejprve uživatel musí vyplnit ip adresu serveru a port, nebo si může vybrat ze starších připojení v tabulce, které aplikace ukládá do lokálního souboru. Kliknutím na tlačítko *Připoj*, se ověří zda daný server je online a pokud ano, ověří se zda je to herní server pro hru, pomocí zprávy `AreYouBang` viz Tabulka 14 v příloze A. Proběhlo-li ověření úspěšně, budou zpřístupněna políčka pro vyplnění loginu, hesla, ale i tlačítka pro registraci a připojení. Jestliže jedno z těchto ověření selže, zobrazí se chybová hláška. Následně se vyplní uživatelské jméno a heslo a přihlásí se do hry. Neexistuje-li uživatel, zobrazí se chybová hláška.

Jestliže uživatel není registrován, nebude moci se přihlásit, musí se tak zaregistrovat v registračním formuláři, viz Obrázek 20b. Po vyplnění formuláře uživatel klikne na tlačítko *Registruj*. Nastane-li v průběhu procesu hlášení chyba, zobrazí se chybová hláška. Nevznikne-li problém, uživatel se bude moci přihlásit do hry.



Obrázek 21: Třídní diagram formuláře pro přihlášení do hry `ServerOption`

Třída `ServerOption` zajišťuje většinu funkcionality formuláře na Obrázku 20a.

Při zobrazení formuláře se provede metoda `initPorts`, která vyhledá, zda existuje soubor, který se nachází ve složce pro aplikaci obsahující historii přihlášení a případně se načte do kolekce `DBports`. Tato kolekce se zobrazí ve formuláři.

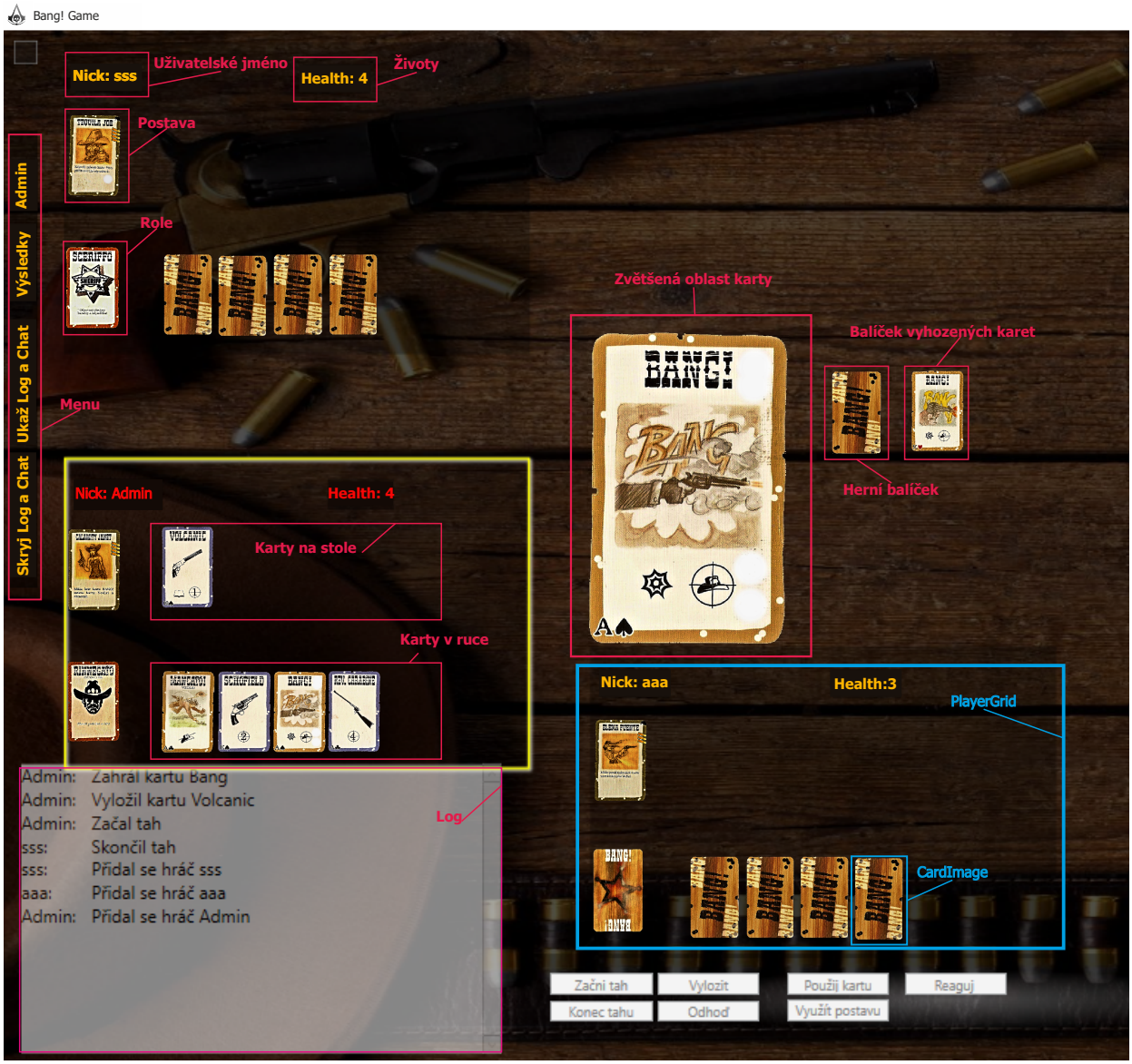
Při vyplňování IP adresy se ověřuje u každého znaku, přes metodu `UpdateUI`, zda se vyplňují čísla v intervalu daném pro IPv4. Pokud je zadáno písmeno nebo výsledné číslo překročí interval, zablokuje se tlačítko *Připoj*. Po úspěšném vyplnění adresy a portu klikne uživatel na tlačítko *Připoj* a provede se metoda `Connect`, která přes instanci třídy `MessageOnClient` zašle na server zprávu `AreYouBangServer` (viz Tabulka 14). Vrátí-li se odpověď stejnou hlavičkou, ověření proběhlo úspěšně a zpřístupní se komponenty popsané výše.

Po vyplnění uživatelského jména, hesla a stisku tlačítka *Přihlaš* se provede metoda `LoginF`, která zašle na server přihlašovací údaje opět přes instanci třídy `MessageOnClient`. Odpoví-li server, že uživatel v databázi na serveru neexistuje, zobrazí se hráči chybové sdělení a bude se muset zaregistrovat.

Pokud uživatel uzavře tento formulář dřív než se přihlásí do hry, objeví se mu dotazující okno, zda si opravdu přeje ukončit tuto aplikaci.

Registrace probíhá také přes instanci třídy `MessageOnClient`. Po vyplnění formuláře (viz Obrázek 20b) a stisknutí tlačítka *Registruj*, se tyto informace odešlou na server, kde se uloží do databáze. Tuto operaci zajišťuje v této třídě metoda `Register`. Pokud vyvstane chyba, objeví se uživateli chybové sdělení.

Hlavní okno



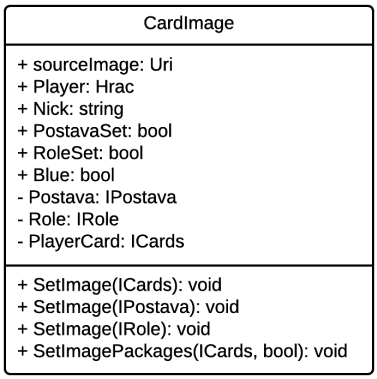
Obrázek 22: Hlavní okno

Ukázku grafického rozhraní hlavního okna vidíme na Obrázku 22. Na hlavním okně se využívá několik uživatelských komponent. Obslužný kód hlavního okna a popis komponent je popsán v následujících podkapitolách.

Komponenty

Hlavní okno využívá dva typy komponent. Tyto komponenty jsou na Obrázku 22 označeny modře:

- `CardImage`
- `PlayerGrid`



Obrázek 23: Třídní diagram komponenty `CardImage`

`CardImage` komponenta (viz Obrázek 23) je vytvořena jako uživatelská komponenta a zajišťuje zobrazení karty posaných v kapitole 3.3.1, navíc obsahuje veškeré informace o kartě, komu karta patří (vlastnost Nick), zda jde o herní kartu atd.

Nastavení této komponenty probíhá přes přetížené metody `SetImage`, pomocí kterých je možno nastavit, o jakou kartu se bude jednat, zda herní kartu, kartu role nebo postav. U herní karty se dále nastavuje, zda je to karta s modrým okrajem. Parametry `PostavaSet` a `RoleSet` jsou ve výchozím stavu nastaveny na hodnotu `false`. Společné vlastnosti, které se nastavují u každé z této přetížené metody jsou `sourceImage`, který ukládá cestu k obrázku a `Player`, zde se ukládá hráč kterému komponenta patří. Při nastavování se u všech metod `SetImage` nastavuje, zda karta bude zobrazená z přední nebo zadní strany, podle toho kterému hráči patří. Pro balíčky je zde samostatná metoda `SetImagePackage`.

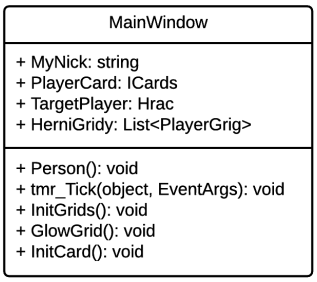
Při kliknutí na komponentu se vyvolá událost `MouseDown`, která předá instanci hlavního formuláře `MainWindow.Instance` příslušné informace, např.: Player, Nick, atd, podle typu nastavení této komponenty, zda se jedná o herní kartu, kartu role (`RoleSet`) nebo postavy(`PostavaSet`). Pokud se jedná o herní karty hráčů, změní jim pozici. U karet v ruce se karta posune nahoru a u karet na stole se posune dolů, aby bylo ukázáno, na kterou kartu hráč kliknul. Pro zvětšení obrázku karet, bez ohledu na nastavení, slouží události

MouseEnter a MouseLeave. Po přejetí myší přes komponentu se obrázek zobrazí v nástroji hlavního okna Image (viz Obrázek 22 s popisem: Zvětšená oblast karty) a po opuštění této komponenty se tento nástroj nastaví na null.

PlayerGrid je komponenta, která zobrzuje veškeré informace od jednoho hráče. Tato komponenta je složena z komonent CardImage. Staticky jsou zde nadefinovány instance CardImage pro roli a postavu, dynamicky se pak tvoří instance pro karty v ruce a karty na stole. Tato komponenta udržuje hráče, kterému tato komponenta náleží.

Karty na stole a v ruce se umísťují do komponent vývojového studia StackPanel (viz Obrázek 22 Karty na stole a Karty v ruce).

MainWindow



Obrázek 24: Třídní diagram hlavního okna MainWindow.

Hlavní okno (viz Obrázek 22) je herním oknem, kde se zobrazují herní karty, výsledky a další funkcionalita, dochází zde k interakci uživatele s aplikací.

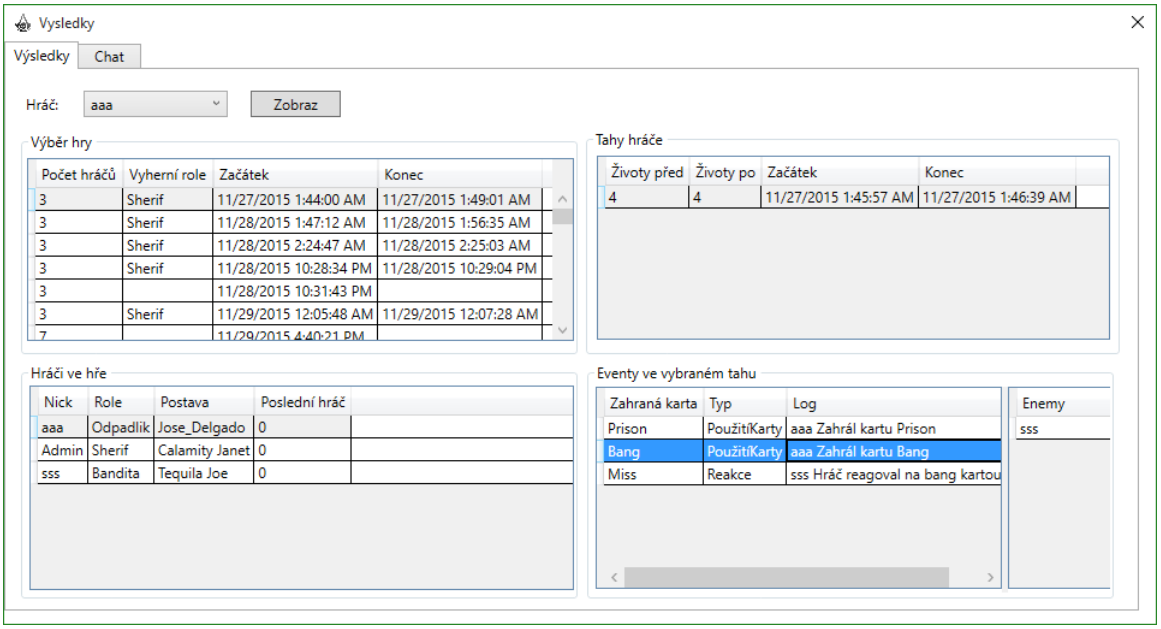
V grafickém rozhraní jsou připraveny již všechny komponenty pro hru od tří do sedmi hráčů a jejich následné rozložení. Rozložení komponent PlayerGrid (viz kapitola 3.4.2 a její podkapitola Komponenty) se nastavuje až při vytvoření hry v závislosti na počtu hráčů.

Ve zdrojovém kódu (viz Obrázek 24), v metodě InitGrids, se komponentám PlayerGrid nastaví příslušné rozložení na herní ploše, přiřadí se příslušný hráč a uloží se do kolekce HerniGridy. Každé instanci komponenty PlayerGrid, která je uložena v kolekci, se nastaví uživatelské jméno, životy, postava a role (viz Obrázek 22). Zde se přidávají nebo odebírají herní karty a komponenty CardImage podle počtu karet v kolekcích hráče (viz kapitola 3.3.2). Toto zajišťuje metoda InitCard.

Pro rozlišení hráče, který je zrovna na tahu, se používá metoda GlowGrid, jenž komponentě PlayerGrid nastaví zelenožlutou záři po obvodu celé komponenty. Dále tato metoda nastavuje i odlišnou barva nicku a životů. Hráč, kterému patří daný klient, bude mít uživatelské jméno a životy vyznačeny červenou barvou, ostatní hráči budou označení žlutou barvou (viz Obrázek 22).

Při vytvoření instance této třídy je spušťěn časovač. Ten aktivuje každou sekundu obslužnou metodu `tmr_Tick` (viz Obrázek 24), která zajišťuje aktuální kolekci hráčů a jejich karet. Během hry registruje události, např.: použití karty Bang nebo konec hry, které během hry nastaly a zobrazuje je hráči, případně mu zpřístupní funkcionalitu, která je vázaná na danou událost. V závěru hry se hráči zobrazí dialog s tím, která role vyhrála. Po akceptování tohoto dialogu dojde k uzavření hlavního okna a uživatel bude vrácen zpět na přihlášení.

Zobrazení výsledků



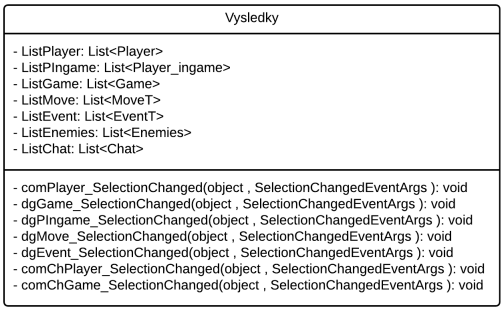
Obrázek 25: Výsledky.

Do okna Výsledky (viz Obrázek 25) se uživatel dostane přes menu (viz Obrázek 22), tlačítkem *Výsledky*. Seznamy, které se zde objevují, dostane uživatel přes třídy, které obstarávají získání dat přes server z databáze na klientovi (viz kapitola 3.4.3).

V záložce *Výsledky* se hráči zobrazí veškeré hry, které odehrál na serveru, přes který je připojený. Uživatel s administrátorskými právy vidí všechny hráče a jejich hry (viz kapitola 3.1). Po výběru hry, v tabulkovém zobrazení, na obrázku označeno *Výběr hry*, se uživateli načtou hráči do v tabulkového zobrazení, na obrázku označenou *Hráči ve hře*, kteří se dané hry zúčastnili. Pokud se jedná o zrovna aktuální hru, jsou role hráčů skryté, a to i pro uživatele s administrátorskými právy. Po výběru příslušného hráče se v tabulkovém zobrazení *Tahy hráče*, zobrazí jednotlivé tahy, které hráč během hry vykonal. Během jednoho tahu může hráč vykonat několik událostí, které jsou zobrazeny v sekci pojmenované na obrázku *Události ve vybraném tahu* s prvním tabulkovým zobrazení. Výběrem příslušného tahu se tyto události zobrazí. Jelikož tyto události mohou obsahovat

útoky na jiné hráče, proto se v této sekci, ve druhém tabulkovém zobrazení pro nepřátele zobrazují i hráči na které byly tyto útoky vedeny.

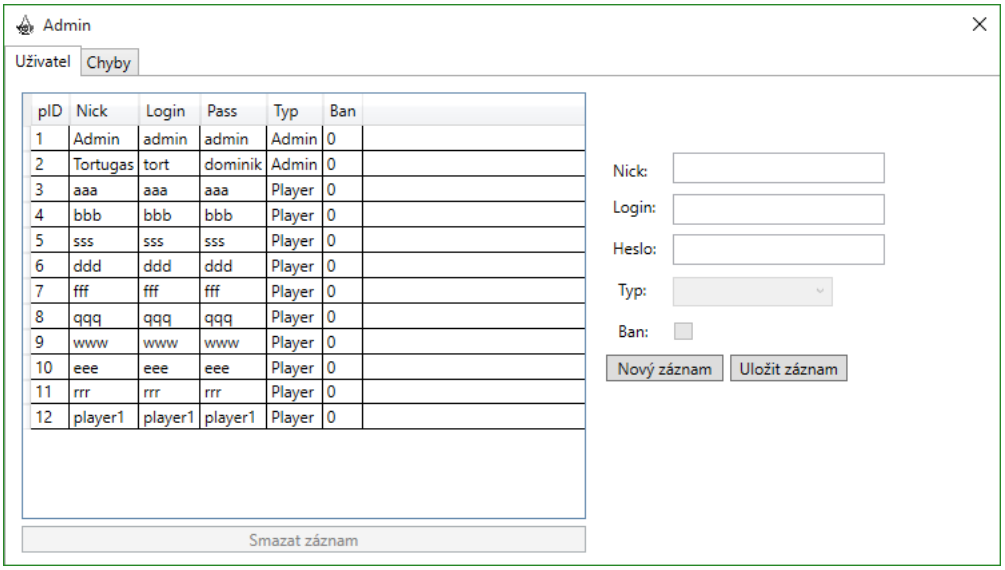
V záložce *Chat* se eviduje konverzace hráčů v jednotlivých hrách. Hráč, ale uvidí jen svůj text, který napsal. Uživatel s právem administrátora se ovšem může podívat i na ostatní hráče.



Obrázek 26: Třídní diagram formuláře třídy `Výsledky`.

Při každé změně v tabulkových zobrazení zmíněných výše se na server zasílá žádost o aktuální data pro tabulku, která tuto změnu vyvolala. Odpověď ze serveru má formu XML (viz kapitola 3.3.5), jelikož se jedná vždy o určitou kolekci dat. Každá z těchto tabulkového zobrazení má metodu `SelectionChanged` (viz Obrázek 26) a tato metoda reaguje na událost změny vybraného řádku.

Administrace

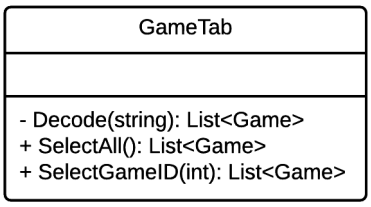


Obrázek 27: Administrace.

Administrace je přístupná z nabídky (viz Obrázek 22 tlačítko *Admin*) a to jen uživatelům s administrátorským právem (viz kapitola 3.3.2). Pro zobrazení dat v tomto okně se obdobně jako ve výsledcích využívají třídy, které obstarávají správu dat z databáze přes server na klienta (viz kapitola 3.4.3).

Administrátor může vytvořit nového hráče, nebo stávající upravit, popřípadě smazat v záložce *Uživatel*. V záložce *Chyby* si může zobrazit chyby, které se reportují z každého klienta na server a poté do databáze. Chyby je možné vyexportovat do souboru.

3.4.3 Přístup k datům z databáze



Obrázek 28: Příklad třídy pro získání dat z databáze na klientovi.

V kapitole 3.3.6 je popsáno ORM pro výměnu dat mezi serverovou aplikací a databází. Vzhledem k tomu, že klient nikdy nepřistupuje k databázi přímo, ale vždy přes server, bylo nutné vytvořit Gatewaey třídy na klientovi. Třídy, které obstarávají práci s požadavky na data, obsahují metody, které vyšlou požadavek na server do ORM na metodu, kterou se týká případný požadavek. Tyto metody mohou pažádat ORM na serveru o DML operace nebo jen získání dat formou dotazu.

Požadavek má vždy tvar:

`{ [hlavička], [tabulka], [operace],[parametr 1], . . . ,[parametr n] }`.

V odpovědi je většinou, pokud se nejedná o DML operace, XML s kolekcí dat, která se zvlášť pro každou tabulku dekóduje ve funkcích `Decode` (viz Obrázek 28). Tato kolekce se vrátí metodě, která inicializovala tuto žádost na databázi.

Příklad 3.6 (viz Obrázek 28)
Uživatel si chce zobrazit všechny hry, co odehrál v okně pro výsledky (viz kapitola 3.4.2, podkapitola Zobrazení výsledků). Po otevření okna s výsledky, vybere hráče, pokud uživatel není administrátor uvidí jen sebe, stiskne tlačítko *Zobrazit*. Toto inicializuje žádost na instanci třídy `GameTab` na klientovi a zavolá metodu `SelectAll`. Tato metoda vyšle na server žádost o data z databáze ke všem hrám, co uživatel odehrál na tomto serveru. Na serveru se zašle požadavek do ORM třídy `GameTable` a metody `SelectAll`. Tato metoda vrátí kolekci objektů `Game`, kterou zabalí do XML a pošle se na klienta. Klient v instanci třídy `GameTab` XML rozbalí pomocí metody `Decode` a kolekci objektů typu `Game` vrátí zpátky do formuláře výsledků, který tuto kolekci objektů `Game` zobrazí v tabulce *Výběr hry*.

3.5 Databáze

Databáze slouží pro ukládání informací z jednotlivých her a také pro získání přístupu do hry. Byla zvolena relační databáze, od Microsoft SQL Server 2014. Relační model databáze je zobrazen na Obrázku 29.

Tabulka *Player* (viz Tabulka 3) slouží pro přihlašování do aplikace. Dále umožňuje určitým uživatelům dostat se do administračního formuláře, tím že se atribut *Typ* nastaví z výchozího *Player* na *Admin*.

Název	Typ	Délka	Klíč	Null	Popis
PlayerID	Integer		PK	ne	Primární klíč tabulky
Nick	Varchar	80		ne	Přezdívka hráče
Login	Varchar	80		ne	Přihlašovací jméno
Pass	Varchar	100		ne	Heslo uživatele
Typ	Varchar	10		ne	Práva uživatele
Ban	Char	1		ne	Možnost dostat se do hry

Tabulka 3: Tabulka *Player*

Na začátku každé hry se vytvoří nový záznam pro v tabulce *Game* (viz Tabulka 4), kde se první vyplní povinné atributy. Po skončení hry se upraví záznam hry a nastaví se mu datum a čas konce hry a vítěznou roli která vyhrála.

Název	Typ	Délka	Klíč	Null	Popis
GameID	Integer		PK	ne	Primární klíč tabulky Game
Count_players	Integer			ne	Počet hráčů ve hře
Begin	DateTime			ne	Začátek hry
End	DateTime			ano	Konec hry
RoleID	Integer		FK	ano	Vítězná role

Tabulka 4: Tabulka *Game*

V další tabulce *PlayerIngame* (viz Tabulka 5) se každému hráči vytvoří záznam pro právě hranou hru, zde se eviduje, jakou měl hráč během této hry roli, postavu a po skončení hry, i kdo tuto hru vyhrál.

Název	Typ	Délka	Klíč	Null	Popis
PIngameID	Integer		PK	ne	Primární klíč tabulky
EndGame	Bollean			ano	Příznak, který hráč vyhrál hru
GameID	Integer		FK	ne	Hra hráče
PlayerID	Integer		FK	ne	Hráč ve hře
RoleID	Integer		FK	ne	Role hráče
CharID	Integer		FK	ne	Charakter hráče

Tabulka 5: Tabulka *PlayerIngame*

Během hry hráč použije několik tahů (každý hráč během kola má jeden tah), které se ukládají do tabulky *Move* (viz Tabulka 6), kde se na začátku tahu vyplní povinné parametry. Na konci tahu se záznam upraví a nastaví se konec tahu a počet životů, které hráči po jeho kole zůstaly.

Název	Typ	Délka	Klíč	Null	Popis
MoveID	Integer		PK	ne	Primární klíč tabulky
StartHealth	Integer			ne	Životy na začátku tahu
EndHealth	Integer			ano	Životy na konci tahu
Begin	DateTime			ne	Začátek tahu
End	DateTime			ano	Konec tahu
PlayerID	Integer		FK	ne	Tah hráče
GameID	Integer		FK	ne	Tah ve hře

Tabulka 6: Tabulka *Move*

Hráč může během jednoho tahu použít několik karet. Použité karty se evidují v tabulce *Event* (viz tabulka 7). Zde se zaznamenává jaký typ karty byl zahrán a jak byla karta použita, zda jako reakce na útok, nebo normální použití během tahu.

Název	Typ	Délka	Klíč	Null	Popis
EventID	Integer		PK	ne	Primární klíč tabulky
Card	Varchar	50		ne	Jméno zahrané karty
Description	Varchar	150		ne	Popis události
MoveID	Integer			ano	Tah, kterému patří tato událost
Type	Varchar	40		ne	Typ zahraní karty

Tabulka 7: Tabulka *Event*

V tabulce *Event* karty, které byly použity jako útok na jiné hráče, se dále eviduje na které hráče byly použity. Jelikož ale útoky mohou být i na více hráčů, evidují se tito hráči v tabulce *Enemies* (viz tabulka 8). Tato tabulka je vazební M:N vazby, mezi tabulkou *Player* a tabulkou *Event*.

Název	Typ	Délka	Klíč	Null	Popis
EventID	Integer		PF	ne	Primární klíč tabulky
PlayerID	Integer		PF	ne	Primární klíč tabulky

Tabulka 8: Tabulka *Enemies*

Pro zobrazení rolí a postav ve výsledcích z databáze slouží dvě tabulky. Tabulka *RoleT* (viz Tabulka 10), kde se evidují role, a tabulka *CharacterT* (viz Tabulka 9), kde se evidují postavy. Tyto tabulky se naplní podle toho, kolik v aplikaci existuje postav a rolí.

Název	Typ	Délka	Klíč	Null	Popis
CharID	Integer		PK	ne	Primární klíč tabulky
Name	Varchar	80		ne	Jméno postavy
Health	Integer			ne	Počet životů postavy

Tabulka 9: Tabulka *CharacterT*

Název	Typ	Délka	Klíč	Null	Popis
RoleID	Integer		PK	ne	Primární klíč tabulky
Name	Varchar	60		ne	Název role

Tabulka 10: Tabulka *RoleT*

V tabulce *Error_report* (viz Tabulka 11) se zaznamenávají chyby, které se vyskytly během chodu aplikace na klientovi. Tyto záznamy si pak uživatel s administrátorskými právy může vyžádat a zobrazit, nebo popřípadě vyexportovat do souboru.

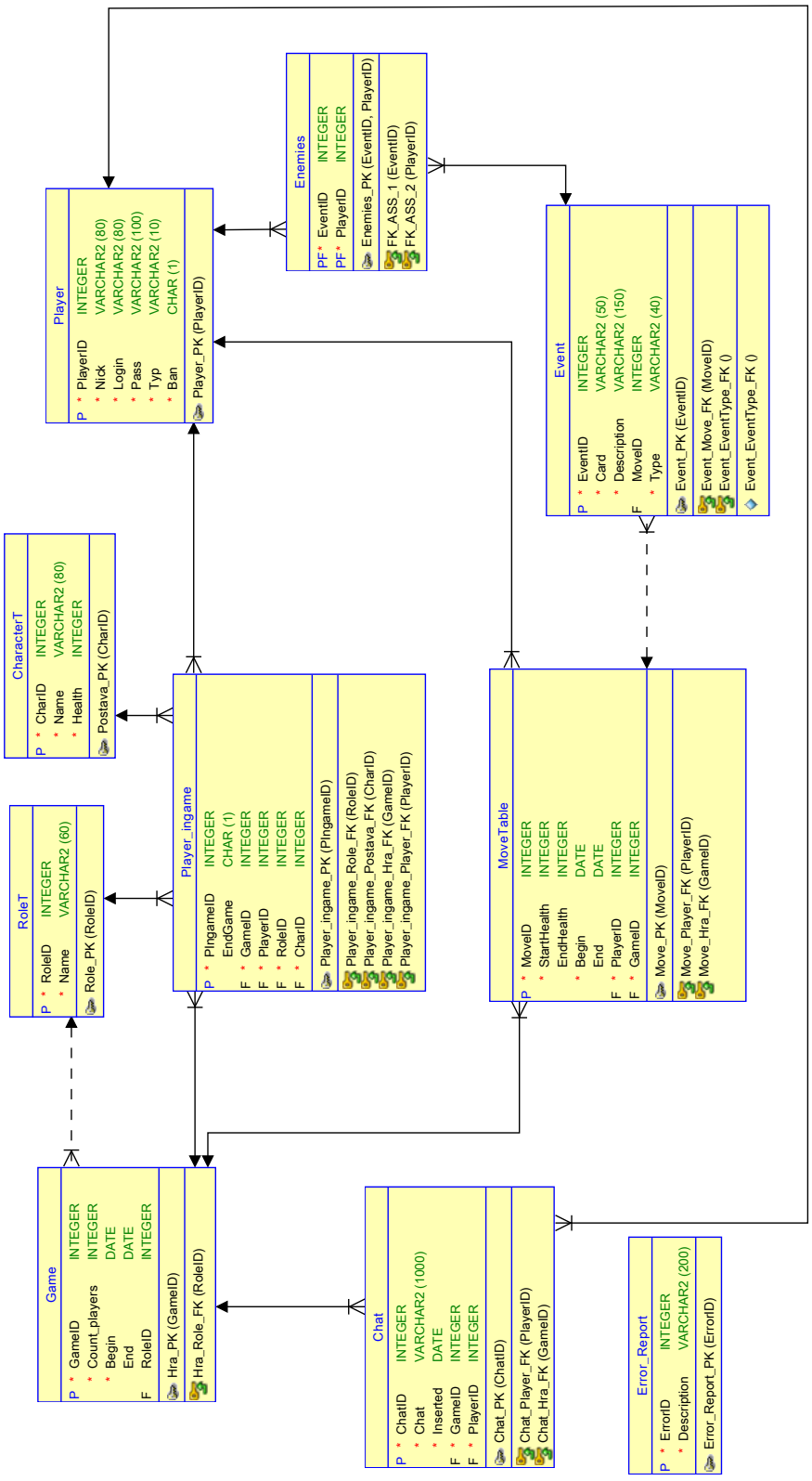
Název	Typ	Délka	Klíč	Null	Popis
ErrorID	Integer		PK	ne	Primární klíč tabulky
Description	Varchar	200		ne	Popis chyby

Tabulka 11: Tabulka *Error_Report*

V tabulce *Chat* (viz Tabulka 12) se zaznamenávájí zprávy, kterými uživatelé během hry komunikují. Uživatel ale v okně kde se zobrazují výsledky, v záložce pro chat, uvidí pouze své zprávy, jedině administrátor si může prohlédnout konverzace všech uživatelů.

Název	Typ	Délka	Klíč	Null	Popis
ChatID	Integer		PK	ne	Primární klíč tabulky
Chat	Varchar	1000		ne	Zpráva napsaná hráčem
Inserted	DateTime			ne	Datum vložení
GameID	Integer		FK	ne	Chat náležící hře
PlayerID	Integer		FK	ne	Chat náležící hráči

Tabulka 12: Tabulka *Chat*



Obrázek 29: Relační model

4 Předchozí řešení

Srovnání s předchozími řešeními bude provedeno na následujících čtyřech řešení, které budou popsány samostatně v následujících podkapitolách. Tyto řešení jsou:

1. KBang!
2. Bang! Video game
3. Předchozí bakalářská práce [9]
4. Předchozí bakalářská práce [10]

Srovnání s prací KBang!

Tato aplikace [5] je mnohem starší, jedná se o jednu z prvních aplikací, které dovolují hrát tuto hru v elektronické podobě. Vývoj této aplikace byl nabídnut jako tzv. Open-Source licence. Aplikace KBang byla vyvíjena v jazyce C a C++ a pro grafické rozhraní byla použita knihotna Qt4. Tato aplikace měla svůj vlastní server, na kterém si uživatelé vytvořili hru. Obsahuje klienta, který zobrazuje průběh hry a je připojen na server. Hra hráči nabízí možnost vytvoření dočasného profilu a nastavení profilového obrázku. Grafické prostředí je koncipováno podobně jak bylo navrženo u naší aplikace. Skládá se z komponent pro hráče, ekvivalent je u mne `PlayerGrid` a pravděpodobně karty jako je u mě ekvivalent `CardImage`.
Ukázka hry na Obrázku 30

Srovnání s prací Bang! Video Game

U tohoto řešení [6] je srovnání pouze z grafické části klienta, jelikož se jedná o komerční verzi, která nemá OpenSource licenci.

Tato aplikace má navrženou vlastní grafiku prostředí např.: karty, postavy atd. Zobrazení životů je zde znázorněno graficky. Tato aplikace je primárně vyvíjena na mobilní platformy, existuje však i jako desktopová verze na operační systém Windows. Pro Android zatím nebyla zveřejněna žádná verze.
Ukázka hry na Obrázku 31

Srovnání s bakalářskou prací od Jana Schovánka

Tato práce [9] byla vytvářena jako webová hra pod jazykem Java, kdežto má práce byla zhotovena jako desktopová hra pod jazykem C#. Základ v této práci tvoří klient, server a databáze podobně jak u této naší aplikace. Data mezi klientem a serverem se přenášejí pomocí JSON, do kterého se zabalí celé kolekce. Podobně jak u naší aplikace, tak v této jsou dvě komponenty **Hráč**, tato komponenta je podobná mé `PlayerGrid` (viz kapitola 3.4.2) a komponenta **Karta**, která je podobná mé komponentě `CardImage`.

Na serveru, stejně jako v naší aplikaci, probíhá logika hry a přístup k databázi. Klient ovšem na rozdíl od naší aplikace běží v nekonečné smyčce pro příjem zpráv, ale v mém provedení běží časovač, který si aktuální data vyžádává.

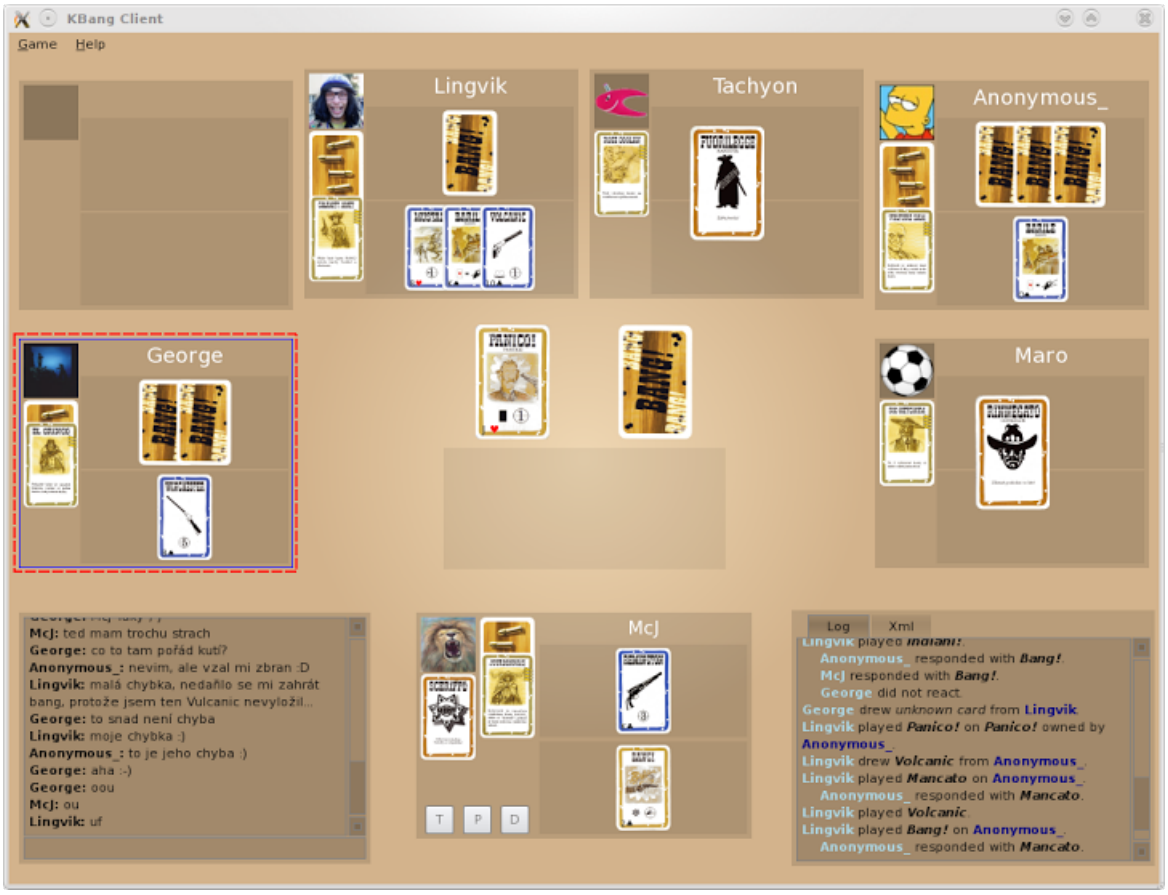
Srovnání s bakalářskou prací od Zdeňka Balickiho

Tato popisována verze [10] je stejně jako ta předchozí (viz podkapitola 4) vyvíjena jako internetová aplikace pod jazykem Java. Rozdíl mezi předchozí prací (viz podkapitola 4) je ten, že tato obsahuje jednoduchou umělou inteligenci a a možnost o rozšíření dalších expanzí této hry. Hra je koncipována podobně jako předchozí a mé řešení. Je zde samostatný server a klienti. Komunikace mezi serverem a klientem probíhá v rámci jednoduchých řetězců. Mé řešení je kombinace XML a jednoduchého řetězce, v závislosti na náročnosti zprávy.

Zhodnocení základních rysů jednotlivých řešení vidíme v Tabulce 13.

Tabulka 13: Tabulka srovnání

	Umělá inteligence	Programovací jazyk	Typ aplikace
KBang!	✗	c++	Desktop
Bang! video game	-	-	Mobilní/ desktop
Bc [10]	✓	Java	Webová
Bc [9]	✗	Java	Webová
Tato aplikace	✗	C#	Desktop



Obrázek 30: Ukázka hry KBang

Obrázek 31: Ukázka hry Bang! Video game

5 Závěr

Cílem této práce byl návrh a implementace společenské karetní hry Bang!. Hra byla navržená jako desktopová aplikace v jazyce C#. Výsledkem je funkční aplikace, kterou je možné reálně používat.

V rámci praktické části, byl úkol analyzovat a navrhnout funkční server, klienta a komunikaci mezi nimi. Pro ukládání výsledků a informacích o hráči byla zvolena databáze, konkrétně databáze od Microsoftu s verzí Microsoft Server 2014.

Pro informace o kartách hráčích, herní logice a přístupu k databázi, byla vytvořená společná knihovna, ze které klient a server přebírají potřebnou funkcionalitu.

Na serveru bylo nutné vytvořit síťovou komunikaci a následné přebírání zpráv zaslaných od klientů. Pro tento účel byl použit TCP protokol a přebírá pouze zprávy jako řetězce. Dále zde bylo nutné vytvořit hru a poté podle zpráv zajistit správné fungování logiky hry a také zpřístupnit komunikaci s databázi.

Na klientovi bylo nutné vytvořit grafické prostředí a rozpoložení komponent po celé herní ploše. Výchozí komponenty ovšem v některých případech nestačily a proto bylo zapotřebí vytvořit vlastní komponenty pro hráče a karty. Dále bylo nutné zajistit síťovou komunikaci se serverem, bez které by klient nefungoval. Tato komunikace zajišťuje klientovi veškerá potřebná data, jako jsou například data z databáze nebo herní průběh.

Poslední důležitou částí této aplikace bylo vytvoření databáze, do které by se ukládaly výsledky, tahy z her, chyby vzniklé v průběhu hry a informace o hráči, jeho přezdívka, uživatelské jméno, heslo a jestli má daný hráč zakázaný přístup do hry.

Do budoucna bych chtěl do aplikace vytvořit umělou inteligenci, která by nebyla závislá na serveru. Co se týče výhod a nevýhod aplikace jsem toho názoru, že výhodou je uzavřený systém, kdy nezkušení uživatelé nemohou přidávat další karty a nová pravidla, ovšem to je současně i nevýhoda, jelikož pro rozšíření této aplikace je nutné umět programovat.

6 Reference

[1] TCP protokol[online]. [cit. 2016-03-17]. Dostupné z:
http://www.cs.vsb.cz/grygarek/POS/lect/PREZENTACE/TCPIP.pdf.

[2] C# [online]. [cit. 2016-03-17]. Dostupné z:
http://ita.vsb.cz/technologie-net-framework.aspx.

[3] C# [online]. [cit. 2016-03-17]. Dostupné z:
http://www.dotnetperls.com/.

[4] C# [online]. [cit. 2016-03-17]. Dostupné z:
https://msdn.microsoft.com/en-us/library/618ayhy6.aspx.

[5] KBang [online]. [cit. 2016-03-17]. Dostupné z:
https://code.google.com/p/kbang/.

[6] Bang!video game [online]. [cit. 2016-03-17]. Dostupné z:
http://www.bangvideogame.com/.

[7] Pravidla Bang! [online]. [cit. 2016-03-17]. Dostupné z:
http://www.bang.cz/cs/bang.html.

[8] Martin Fowler: Patterns of Enterprise Application Architecture 2003. [cit. 2016-03-17]

[9] Jan Schovánek: Online verze karetní hry BANG! 2011. [cit. 2016-03-17]

[10] Zdeněk Balicki: Hra Bang! v prostředí internetu 2014. [cit. 2016-03-17]

A Tabulka zpráv

Tabulka 14: Tabulka zpráv

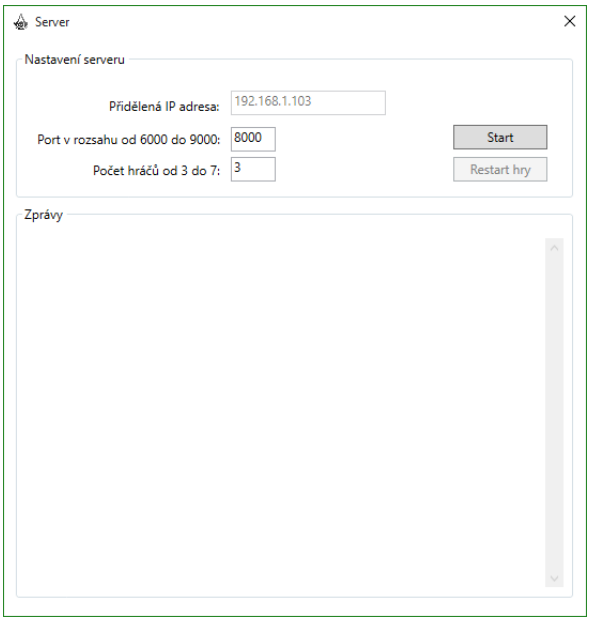
Hlavička zprávy	Odchozí parametry	Parametry odpovědi	Popis
NewGame	Přezdívka		Vytvoří nového hráče na serveru.
AllPlayers		XML	Požadavek o získání aktuální situace hry.
Vyloz	ID karty, ID hráče		Vyloží kartu hráče, podle jeho id, z kolekce karet v ruce do kolekce karet nastole podle id karty.
UseCard	ID karty, ID hráče, ID nepřátelského hráče, ID karty nepřátelského hráče		Použije kartu hráče na nepřátelského hráče. ID nepřátelské karty není povinné, jen v případě, že se karta nepřátelskému hráči odebírá.
ReactToCard	ID karty, ID hráče, ID nepřátelského hráče, typ akce nepřátelského hráče na kterou se reaguje		Reaguje na útok od nepřátelského hráče. ID karty není nutný parametr v případě, že hráč nemá k reakci kartu, pošle se místo toho -1, která značí neexistenci karty.
useChar	ID karty, ID hráče, ID nepřátelského hráče, typ akce nepřátelského hráče na kterou se reaguje		Použití schopnosti postavy. Obsah zprávy se odvíjí od typu schopnosti postavy. Jediný povinný parametr je ID hráče.
move	ID hráče		Provede první tah v kole hráče.
EndMove	ID hráče		
overNick	Přezdívka hráče	true/false	Ověří zda zadaný nick není používán v době hry na serveru.
GameOverPlayer	ID hráče, ID nepřátelského hráče	true/false	Při smrti hráče, se tento hráč označí, že umřel. Pokud měl roli sherifa, ukončí se hra podle pravidel. ID nepřátelského hráče není nutné, vyplňuje se pouze pokud smrt byla zaviněná cizím hráčem.

Tabulka 14: Tabulka zpráv

Hlavička zprávy	Odchozí parametry	Parametry odpovědi	Popis
ClosedWin			Při ukončení okna a pokud hráč ještě neumřel se hra ukončí. Pokud hráč, který zavřel okno už umřel hra pokračuje dále.
Odhod	ID karty, ID hráče		Odhodí kartu hráče na základě ID.
Dynamit	ID hráče		Otestuje zda dynamit vybouchnul, pokud ne posune se karta dynamitu dalšímu hráči.
Prison	ID hráče, ID karty	true/false	Otestuje hráče, který je ve vězení, zda se mu podařilo, nebo nepodařilo, dostat se z vězení.
GetPlayers		Počet hráčů	Získání maximálního počtu hráčů pro hru na serveru.
Obsazeno		Počet hráčů	Získání obsazenosti serveru.
AreYouBangServer			Dotaz na herní server.
Login	Uživatelské jméno, Heslo	Přezdívka hráče, typ práv	Přihlásí uživatele do aplikace.
Register	Přezdívka hráče, Uživatelské jméno, Heslo	true/false	Zaregistruje uživatele na server.
GetResults	Jméno tabulky, Typ operace, Parametry dle operace	XML	Získání dat z databáze, nebo jejich upravení, smazání a vytvoření.
WriteError	Popis chyby		Zašle záznam chyby na server a následně ho uloží do databáze.
ChatSend	Přezdívka hráče, Zpráva hráče	XML	Odešle zprávu chatu hráče na server. Klient obdrží aktuální chat všech hráčů.
Admin	Typ operace, Parametry dle operace	OK	Upraví, smaže nebo vloží záznamy v tabulce Player.
EGTotal			Ukončí hru.

B Návod

Před spuštěním serveru je třeba nastavit připojovací řetězec na databázi, který se nachází v textovém souboru spolu se serverem. Tento nový řetězec musí být na prvním řádku a musí být zakončen tímto symbolem "|". Po spuštění serveru se už jen nastaví port na kterém server bude přístupný a počet hráčů, kteří budou na tomto serveru moct hrát (viz Obrázek 32). Po stisku na tlačítko *Start* se spustí server. Pro případ, kdyby se hra na serveru po ukončení neresetovala je zde tlačítko *Restart hry*, které vynuluje hru.



Obrázek 32: Server

Po spuštění klienta se zobrazí okno (viz Obrázek 20a), kde se vyplní příslušná IP adresa serveru a port. Poté se klikne na tlačítko *Připoj* a následně se vyplní přihlašovací údaje. Pokud není uživatel registrován musí se zaregistrovat přes registrační formulář (viz Obrázek 20b). Po přihlášení se uživateli zobrazí herní okno a po připojení všech hráčů začne hra.

C Příloha na CD/DVD

- *Bin* – složka se spustitelným programem.
- *Database* – složka se zálohou databáze a scripty pro vytvoření databáze.
- *Doc* – složka s textovou částí práce.
- *Src* – složka s projektem.
- *readme.txt* – pokyny pro správné spuštění aplikace.